# IPS Signature Syntax Guide

IPS Signature Syntax Guide

May 22, 2014

00-108-229429-20140522



| | |
|---|---|
| Fortinet Document Library | docs.fortinet.com |
| Fortinet Video Library | video.fortinet.com |
| Fortinet Knowledge Base | kb.fortinet.com |
| Customer Service & Support | support.fortinet.com |
| Training Services | training.fortinet.com |
| FortiGuard | fortiguard.com |
| Document Feedback | techdocs@fortinet.com |

# Table of Contents

# Table of Figures

# List of Tables

# Change Log

| Date | Change Description |
|---|---|
| 2012-05-05 | Initial release. |
| 2013-01-13 | Add new options bring by IPS engine 2.153. |
| 2013-11-08 | Updated data_size HTTP_HOST part. |
| 2014-01-07 | Add new keywords pset/lasttag and B.4. |
| 2014-01-17 | Updated to current template. |
| 2014-05-22 | Minor document updates. |
| | |
| | |

# Quick Reference

The following table provides a quick reference for the format and syntax of signatures for IPS Engine 4.0.

**Table 1:** IPS signature quick reference

| Field | Values | String |
|---|---|---|
| `--name` | `<text string>;` (max 64 char) | `--name "IBM.Domino.iNotes.Foldername.Buffer.Overflow"` |
| `--protocol` | `<name>;` or `<number>;` | `--protocol tcp;` |
| | | `--protocol 2;//IGMP protocol` |
| `--ip_id` | `<number>;` | `--ip_id 32212;` |
| `--ip_tos` | `<number>;` | `--ip_tos 4;` |
| `--ip_ttl` | `<number>;, > <number>;, < <number>;` | `--ip_ttl 4;` |
| | | `--ip_ttl >4;` |
| | | `--ip_ttl <4;` |
| `--ip_option` | `rr` - record route | `--ip_option rr;` |
| | `eol` - end of list | `--ip_option eol;` |
| | `nop` - no operation | `--ip_option nop;` |
| | `ts` - Internet timestamp | `--ip_option ts;` |
| | `sec` - security | `--ip_option sec;` |
| | `lsrr` - loose source routing | `--ip_option lsrr;` |
| | `lsrre` - LSRRE | `--ip_option lsrre;` |
| | `ssrr` - strict source routing | `--ip_option ssrr;` |
| | `satid` - stream ID | `--ip_option satid;` |
| | `any` - any IP option | `--ip_option any;` |
| `--same_ip` | N/A | `--same_ip;` |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --src_addr | [!]<IP address>;<br>format:<br>x.y.z.u<br>x.y.z.u/n<br>x.y.z.u:n<br>ab:cd:ef:gh:ij:kl:mn:op<br>ab:cd:ef::mn:op<br>! = exclude<br>multiple addresses: [address1],[address2]… | --src_addr 10.10.10.1 |
| | | --src_addr 10.10.10.0/24 |
| | | --src_addr 10.10.10.0:24 |
| | | --src_addr XX12:YY34:ZZ56:0000:0000:AA78:BB90 |
| | | --src_addr XX12:YY34:ZZ56::AA78:BB90 |
| | | --src_addr !10.10.10.1 |
| | | --src_addr [10.10.10.1],[10.10.10.2],[10.10.10.3] |
| --dst_addr | [!]<IP address>;<br>format:<br>x.y.z.u<br>x.y.z.u/n<br>x.y.z.u:n<br>ab:cd:ef:gh:ij:kl:mn:op<br>ab:cd:ef::mn:op<br>! = exclude<br>multiple addresses: [address1],[address2]… | --dst_addr 10.10.10.1 |
| | | --dst_addr 10.10.10.0/24 |
| | | --dst_addr 10.10.10.0:24 |
| | | --dst_addr XX12:YY34:ZZ56:0000:0000:AA78:BB90 |
| | | --dst_addr XX12:YY34:ZZ56::AA78:BB90 |
| | | --dst_addr !10.10.10.1 |
| | | --dst_addr [10.10.10.1],[10.10.10.2],[10.10.10.3] |
| --ipver | <version number>; | --ipver 6; //detect IP version 6 packets |
| --ipv6h | <next header>; | --ipver 6; --ipv6h 0; //detect IPv6 packets which next header is hop-by-hop option |
| | | --ipver 6; --ipv6h 58; --protocol icmp; --icmp_type 135; --icmp_code 0 //detect ICMPv6 packets which type value is 135 and code value is 0. |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --ip.total_length | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --ip.total_length >= 402; |
| --ip.id | | --ip.id & 0xff = 0x37; |
| --ip.ttl | | --ip.ttl in [64,65]; |
| --ip.checksum | | --ip.checksum != 0xff; |
| --ip6.payload_length | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --ip6.payload_length > 40; |
| --ip6.next_header | | --ip6.next_header in [1,2]; |
| --ip6.hop_limit | | --ip6.hop_limit < 0x4f; |
| --ip[offset] | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --ip[2] >= 402,word; |
| | | --ip[4] & 0xff = 0x37,word; |
| --ip6[offset] | <operator><value><word size><endianness>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --ip6[4] > 40,word; |
| --src_port | <number>;<br>! = exclude<br>: = range | --src_port 1000:; // >=1000 |
| --dst_port | <number>;<br>! = exclude<br>: = range | --dst_port 100:200; // >=100 and <=200 |
| --seq | <number>[,relative];<br>=, number[,relative]<br>>, number[,relative]<br><, number[,relative]<br>!, number[,relative] | --seq <,12345; |
| | | --seq !, 12345; |
| --ack | <number>; | --ack 12345; |
| --tcp_flags | <!*FSRPAU120>[,<FSRPAU120>];<br>S = SYN<br>A = ACK<br>F = FIN<br>R = RST<br>U = URG<br>P = PSH<br>1 = (reserved bit 1)<br>2 = (reserved bit 2)<br>0 = no TCP flags set | --tcp_flags 0,12; |
| | | --tcp_flags !SAFRUP,12; |
| | | --tcp_flags S,12; |
| | | --tcp_flags S+; |
| | | --tcp_flags *SAFRUP12; |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --window_size | [!]<number>;<br>[!]<0x number>;<br>0x indicates Hexadecimal | --window_size 1000;<br>--window_size !0x1000; |
| --tcp.src_port | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --tcp.src_port in [1111,2222]; |
| --tcp.dst_port | | --tcp.flags & 0x0f = 0x6 |
| --tcp.seq | | --tcp.any_option = 0x6052, dword; //iterate over all options |
| --tcp.ack | | |
| --tcp.flags | | |
| --tcp.window_size | | |
| --tcp.checksum | | |
| --tcp.urgent | | |
| --tcp.any_option | | |
| --tcp[offset] | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --tcp[20] &0xF0 = 0x30 |
| --udp.src_port | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --udp.src_port in [1111,2222]; |
| --udp.dst_port | | |
| --udp.length | | |
| --udp.checksum | | |
| --udp[offset] | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --udp[20] &0xF0 = 0x30 |
| --icmp_type | <number>; | |
| --icmp_code | <number>; | |
| --icmp_id | <number>; | |
| --icmp_seq | <number>; | |
| --icmp.code | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --icmp.code in [1,2]; |
| --icmp.type | | |
| --icmp.checksum | | |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --icmp[offset] | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --icmp[1] in [1,2]; |
| --icmp6.code<br>--icmp6.type<br>--icmp6.checksum | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --icmp6.code = 135; |
| --icmp6[offset] | <operator><value>;<br>available operators: =, !=, >=, <=, &, \|, ^, in | --icmp6[0] = 135; |
| --pattern | <string>; [--context c;] [--no_case;] [--distance n[,<refer>];] [--within n[,<refer>];] | --pattern "/level/";<br>--pattern "\|E8 D9FF FFFF\|/bin/sh";<br>--pattern !"\|20\|RTSP/"; |
| --pcre | [!]"/<regular expression>/[<operator>]";<br>i = case insensitive<br>s = include new lines in the dot (.) meta character<br>m = match - See documentation for more information.<br>x = white space ignored<br>A = pattern must match only at start of buffer<br>E = set $ to match only at end of subject string<br>G = inverts greediness of quantifiers | --pcre "/\sLIST\s[^\n]*?\s\{/smi"; |
| --context | PACKET | --context PACKET |
| | PACKET_ORIGIN | --context PACKET_ORIGIN |
| | URI | --context URI |
| | HEADER | --context HEADER |
| | BODY | --context BODY |
| | BANNER | --context BANNER |
| | HOST | --context HOST |
| | FILE | --context FILE |
| --no_case | | --no_case; |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --distance | <range>[,<refer>];<br>reference options:<br>MATCH<br>PACKET<br>CONTEXT<br>REVERSE<br>LASTTAG | --pattern !"\|0a\|"; --within 100,match; |
| --distance_abs | | --pattern !"\|0a\|"; --within_abs 100,match; |
| --within | | --pattern "/disp_album.php?"; --context uri; --no_case; --within 50,context; |
| --within_abs | | --distance 10,packet,reverse; --within 5,packet; |
| --byte_jump | <*\|bytes>,<offset>[,<multiplier>[,<modifiers>]];<br><*\|bytes>: number of bytes from payload to be converted (ASCII or binary)<br><offset>: specifies starting point<br><multiplier>: optional numeric value<br><modifiers>: relative, big, little, string, hex, dec, oct, align | --byte_jump 4,0,relative; |
| | | --byte_jump 4,20,relative,align; |
| | | --byte_jump 4,0,4,relative,little; |
| --byte_test | <*\|bytes>,<op>,<value>,<offset>[,<multiplier>[,<modifiers>]];<br><*\|bytes>: number of bytes from payload to be converted (ASCII or binary)<br><op>: >, <, =, !, &, ~, ^<br><value>: specifies value to be compared with<br><offset>: specifies starting point<br><multiplier>: optional numeric value<br><modifiers>: relative, big, little, string, hex, dec, oct, align | --byte_test 4,>,3536,0,relative; |
| | | --byte_test 4,>,0x7FFF,4,relative; |
| | | --byte_test 4,>,$PKT_SIZE,4,relative; |
| | | --byte_test 4,>,$PKT_SIZE-4,2,relative; |
| --service | <service name>; | --service HTTP; |
| | | --service DNS; |
| --flow | <direction>;<br>options:<br>FROM_CLIENT<br>FROM_SERVER<br>BI_DIRECTION<br>REVERSED | --flow from_client; |
| | | --flow bi_direction; --dst_port 123; //match if src or dst port is 123 |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --tag | <op>,[!]<name>;<br><op>: set, pset, clear, toggle, test, reset, quiet<br>[!]: only allowed in TEST operations | --tag quiet; --tag set,Tag.Rsync.Argument; |
| | | --tag quiet; --tag clear,tag.login; |
| | | --tag test,Tag.Rsync.Argument; |
| | | --tag test,!DHTML.EDIT.CONTROL.CLSID; |
| --parsed_type | <type><br>available values:<br>SSL_PCT<br>SSL_V2<br>SSL_V3<br>TLS_V1<br>TLS_V2<br>SOCK4<br>SOCK5<br>HTTP_GET<br>HTTP_POST<br>HTTP_CHUNKED | F-SBID( --name "FrontPage.Fp30reg.Chunked.Overflow"; --protocol tcp: --service HTTP; --flow from_client; --pattern "POST"; --context uri; --distance 0,context; --within 5,context; --pattern "/_vti_bin/_vti_aut/fp30reg.dll"; --context uri; --no_case; --distance 0; --pattern "Transfer-Encoding"; --context header; --no_case; --pattern "chunked"; --context header --no_case; --distance 1; ) |
| --dhcp_type | <value>;<br>1 = DISCOVER<br>2 = OFFER<br>3 = REQUEST<br>4 = DECLINE<br>5 = ACK<br>6 = NAK<br>7 = RELEASE<br>8 = INFORM<br>9 = RELAY_CLIENTREQUEST<br>10 = RELAY_SERVERREPLY | --dhcp_value 1 |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| --data_size | [op]<value>[,field];<br>if [op] is not present, "=" is assumed.<br>other [op] values: >, <, =<br><value> = decimal number<br>[field] options:<br>PAYLOAD<br>URI<br>HEADER<br>BODY<br>HTTP_CONTENT<br>HTTP_CHUNK<br>HTTP_HOST<br>SMTP_BDAT<br>SMTP_XEXCH50 | --data_size <128; |
| | | --pattern "/admin_/help/";<br>--context uri;<br>--no_case; --data_size >1024,uri; |
| | | --parsed_type HTTP_POST;<br>--pattern "nsiislog.dll";<br>--context uri;<br>--no_case; --data_size >1000,http_content; |
| --rate | --rate <count>,<duration>[,limit];<br><count>: number of matches needed before log entry is generated<br><duration>: time over which matches are counted, in seconds<br>[limit]: counts in strict time rather than averaging over a period of time | F-SBID( --name DHCP.FLOOD;<br>--protocol UDP;<br>--service DHCP;<br>--dhcp_type 1; --rate 100,10; --track DHCP_CLIENT;) |
| --track | --track <keyword>;<br><keyword> options:<br>SRC_IP<br>DST_IP<br>DHCP_CLIENT<br>DNS_DOMAIN<br>DNS_DOMAIN_AND_IP | |
| --log | --log <keyword>;<br><keyword> options:<br>DHCP_CLIENT<br>DHCP_CC_ID<br>DNS_QUERY | --log DHCP_CLIENT; |
| --data_at | <number>[,relative]; | --data_at 100,relative; |
| --rpc_num | <number1>[,<number2>[,<number3>]];<br><number1>: application<br><number2>: version<br><number3>: procedure<br>(* can be used to represent any value) | --rpc_num 100221,*,*; |
| | | --rpc_num 100005,2,*; |
| | | --rpc_num 100000,*,4; |

**Table 1:** IPS signature quick reference (continued)

| Field | Values | String |
|---|---|---|
| `--file_type` | `<class>;`<br>subtypes:<br>`COMPRESS`: arj, bzip, bzip2, cab, gzip, lzh, lzw, rar, rpm, tar, upx, zip<br>`IMAGE`: gif, gif87a, gif89a, jpg, jpeg, png<br>`SCRIPT`: bat, css, hta, vba, vbs, genscript, javascript, perlscript, shellscript, wordbasic<br>`VIDEO`: avi, mpeg<br>`AUDIO`: mp3<br>`STREAM`: stream<br>`MSOFFICE`: msoffice, ppt<br>`PDF`: pdf<br>`FLASH`: flash<br>`EXE`: com, dll, exe<br>`HTML`: html<br>`XML`: xml, wordml<br>`UNKNOWN`: unknown, ActiveMIME, aim, form, hlp, mime, txt | `--file_type PDF;`<br>`--file_type EXE;` |
| `--crc32` | `<checksum>,<file_length>;` | `--crc32 3174B5C8,20480;` |

# Introduction

## Overview

The purpose of this document is to provide details about the format and syntax of signatures for IPS Engine 4.0. Signatures are one of the most critical parts of IPS. All signatures written should comply strictly with the format defined in this document.

## Introduction

Fortinet's IPS Signatures, or rules, falls into two categories: official and custom. This guide only discusses custom rules. Official rules are made by Fortinet analysts and are distributed in the official release packages. Custom user rules are made by our customers for their own use. It should be noted that the maximum allowed length of a custom user signature is 1024 bytes, while the maximum length of an official release signature is 4096 bytes.

The Fortinet IPS Engine uses a simple, lightweight signature definition language. An IPS signature is made up of a type header and a series of option/value pairs, as shown in the format below:

```
TYPE( <option1 value1>; <option2 value2>; ... )
```

- The `TYPE` for normal signatures is "`F-SBID`".
- An option starts with "--", followed by the option name, a space and the option value, and ends with a semicolon ";".
- Option names and keywords are case insensitive.
- Some options do not need a value.

The options are divided into four categories based on their purpose:

- ***Basic options***: used to identify and control a signature, but not related to packet inspection. These options are not addressed in this guide.
- ***Protocol related options***: used to detect IP/ICMP/UDP/TCP protocol headers.
- ***Payload related options***: used to detect the packet payload.
- ***Special options***: options used for all other purposes.

In the following sections, we will discuss each of these option categories in detail. The tables in "Signature options" on page 50 contain links to more information on all of the options.

# Protocol Related Options

## name

The `name` keyword is used to provide a signature name that is displayed in the GUI and the CLI. It is mandatory in all rules. The `name` should only contain printable characters without spaces, and it should be in a user friendly format. Usually we use the following format for a signature name:

```
companyname.productname.vulnerablefile.vulnerabilityname
```

**Notes**

- The `name` should be enclosed with double quotation marks.
- The maximum length of a signature `name` is 64 characters.
- The `name` must be unique for signatures with different `vuln_ids`.

**Format**

```
--name <"text string">;
```

**Example**

```
--name "IBM.Domino.iNotes.Foldername.Buffer.Overflow";
```

## protocol

The keyword `protocol` is used to specify the type of protocol that is associated with the signature. It should be noted that `protocol` only supports protocols that have a protocol number associated with them. That is, the protocol can be identified from an IPv4 "`protocol`" field or an IPv6 "`next header`" field. This is a mandatory field for all signatures.

Besides TCP and UDP, protocols can also be specified by their protocol numbers.

**Format**

```
--protocol <name or number>;
```

**Examples**

```
--protocol tcp;
--protocol 2; //IGMP protocol
```

## IP header options

The details of the IPv4 packet format are shown in Figure 1. The IPv6 header format is shown in Figure 2 on page 19.

**Figure 1:** IPv4 packet format

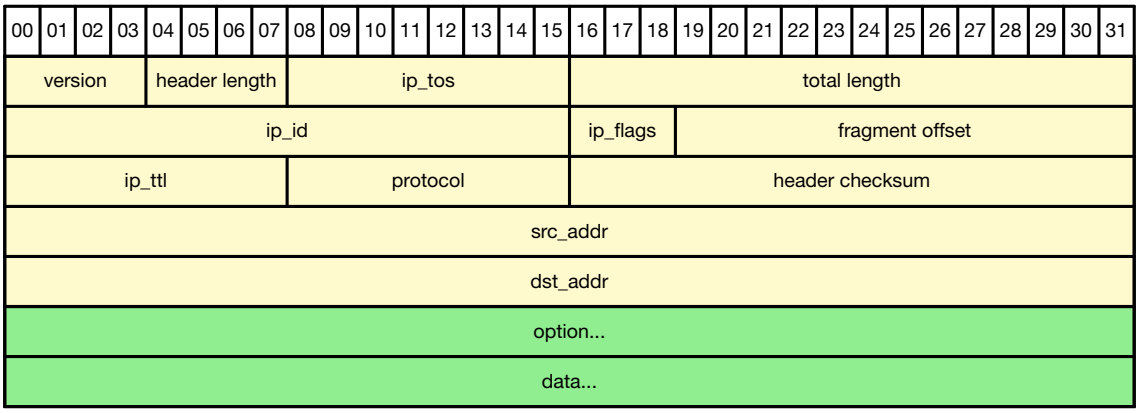
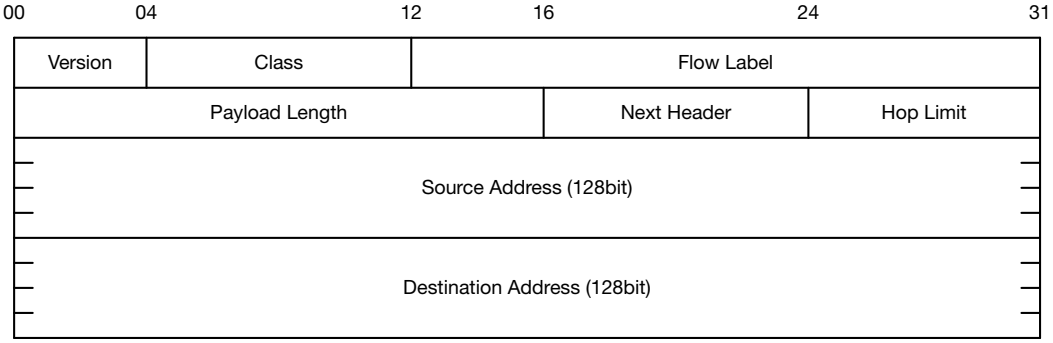



**Figure 2:** IPv6 header format




The following tests are available to check the properties of the IP header:

**Table 2:** IP header property tests

| Keyword | Value | Description |
|---|---|---|
| `--protocol` | `<tcp/udp/icmp/number>;` | Check the protocol type name or number in the IP header.<br>Examples:<br>`--protocol icmp;`<br>`--protocol tcp;`<br>`--protocol udp;`<br>`--protocol 2;`<br>Note: `protocol ip` is not allowed. |
| `--ip_ id` | `<number>;` | Check the IP ID field for a specific value.<br>Example:<br>`--ip_id 32212;` |

**Table 2:** IP header property tests (continued)

| Keyword | Value | Description |
|---|---|---|
| `--ip_tos` | `<number>;` | Check the IP TOS field for a specific value.<br>Example:<br>`--ip_tos 4;` |
| `--ip_ttl` | `<number>;`<br>`><number>;`<br>`<<number>;` | Check the IP time-to-live field value.<br>Example:<br>`--ip_ttl <4;` |
| `--ip_option` | `<option>;` | Check the IP options. The following values can be tested:<br>`rr` Record route.<br>`eol` End of list.<br>`nop` No operation.<br>`ts` Internet timestamp.<br>`sec` Security.<br>`lsrr` Loose source routing.<br>`lsrre` LSRRE.<br>`ssrr` Strict source routing.<br>`satid` Stream ID.<br>`any` Any IP options.<br>Examples:<br>`--ip_option ts;`<br>`--ip_option any;` |
| `--same_ip` | N/A | Check whether `src_addr` is the same as `dst_addr`.<br>Example:<br>`--same_ip;` |
| `--src_addr` | `<IP address>;` | Check the source IP address.<br>The IP address can be in the following formats:<br>`x.y.z.u`<br>`x.y.z.u/n`<br>`x.y.z.u:n`<br>`ab:cd:ef:gh:ij:kl:mn:op`<br>`ab:cd:ef::mn:op`<br>The prefix '`!`' means exclude the addresses. Multiple IP addresses should be between square brackets, '`[ ]`', separated by '`,`'.<br>Examples:<br>`--src_addr !10.10.10.1;`<br>`--src_addr 10.10.10.0:24;`<br>`--src_addr fde0:6477:1e3f::1:b9;` |

**Table 2:** IP header property tests (continued)

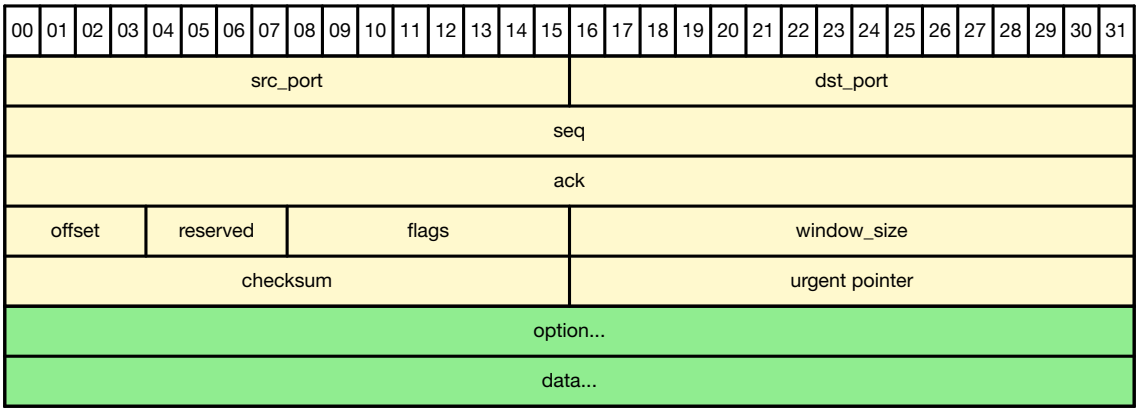
| Keyword | Value | Description |
|---|---|---|
| `--dst_addr` | `<IP address>;` | Check the destination IP address.<br>Refer to `src_addr` for the address format.<br>Examples:<br>`--dst_addr 10.10.10.0/24;`<br>`--dst_addr ![10.10.10.0/24,10.10.20.0:24];`<br>`--dst_addr fde0:6477:1e3f::2:ba;` |
| `--ip_ver` | *<version_number>* | The IP version number.<br>Examples:<br>`--ipver 6; //detect IP version 6 packets` |
| `--ipv6h` | *<Next header>* | A decimal number which is the next header value in IPv6 header.<br>Example:<br>`--ipver 6; --ipv6h 0; //detect IPv6 packets for which the next header is a hop-by-hop option.`<br>`--ipver 6; --ipv6h 58; --protocol icmp; --icmp_type 135;`<br>`--icmp_code 0; //detect ICMPv6 packets for which the type value is 135 and the code value is 0.`<br>Note:<br>`ipv6h` can only be used when `ipver 6` is present. |
| `--ip.total_length`<br>`--ip.id`<br>`--ip.ttl`<br>`--ip.checksum` | *operator value* | Check fields `total_length`, `id`, `ttl`, and `checksum` in IPv4 header. The syntax is:<br>`--ip.[Decorations] operator value;`<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Examples:<br>`--ip.total_length >= 402;`<br>`--ip.id & 0xff = 0x37;`<br>`--ip.ttl in [64,65];`<br>`--ip.checksum != 0xff;` |

**Table 2:** IP header property tests (continued)

| Keyword | Value | Description |
|---|---|---|
| `--ip6.payload_length`<br>`--ip6.next_header`<br>`--ip6.hop_limit` | *operator value* | Check fields `payload_length`, `next_header`, and `hop_limit` in IPv6 header. The syntax is:<br>`--ip6.[Decorations] operator value;`<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Examples:<br>`--ip6.payload_length > 40;`<br>`--ip6.hop_limit < 0x4f;`<br>`--ip6.next_header in [1, 2];` |
| `--ip[offset]` | *operator value* | Access any fields in IPv4 header in a freelance mode. The syntax is:<br>`--ip[offset] operator value [, word size] [, endianness]`<br>Both word size and endianness are optional. By default, the engine uses big endian and BYTE respectively.<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Examples:<br>`--ip[2] >= 402,word;`<br>`--ip[4] & 0xff = 0x37,word;` |
| `--ip6[offset]` | *operator value* | Access any fields in IPv6 header in a freelance mode. The syntax is:<br>`--ip6[offset] operator value [, word size] [, endianness]`<br>Both word size and endianness are optional. By default, the engine uses big endian and BYTE respectively.<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--ip6[4] > 40,word;` |

## TCP header options

**Figure 3:** TCP packet format




The following options are available to check the properties of the TCP header:

**Table 3:** Check TCP header properties

| Keyword | Value | Description |
|---|---|---|
| `--src_port` | `[!]<number>;`<br>`[!]:<number>;`<br>`[!]<number>:;`<br>`[!]<number>:<number>;` | Check the source port number or range.<br>• The prefix `!` means exclude.<br>• `:` is used to indicate a range.<br>Example:<br>`--src_port 1000:; // >=1000` |
| `--dst_port` | `[!]<number>;`<br>`[!]:<number>;`<br>`[!]<number>:;`<br>`[!]<number>:<number>;` | Check the destination port number or range.<br>• The prefix `!` means exclude.<br>• `:` is used to indicate a range.<br>Example:<br>`--dst_port 100:200; // >=100 and <=200` |
| `--seq` | `<number>[,relative];`<br>`=,<number>[,relative];`<br>`>,<number>[,relative];`<br>`<,<number>[,relative];`<br>`!,<number>[,relative];` | Check the TCP sequence number value or range.<br>• `relative` indicates the value is relative to the initial sequence number of the TCP session.<br>• `=` means equal.<br>• `>` means more than.<br>• `<` means less than.<br>• `!` means not equal.<br>• No prefix means equal.<br>Examples:<br>`--seq <,12345;`<br>`--seq !,12345;` |

**Table 3:** Check TCP header properties (continued)

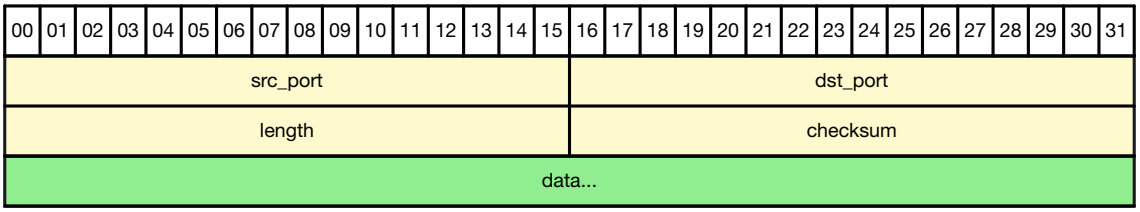
| Keyword | Value | Description |
|---|---|---|
| `--ack` | `<number>;` | Check the TCP acknowledge number for a specific value.<br>Example:<br>`--ack 12345;` |
| `--tcp_flags` | `<!*+FSRPAU120>[,<FSRPAU120>];` | Specify the TCP flags to match in a TCP packet.<br>`S` (SYN)<br>`A` (ACK)<br>`F` (FIN)<br>`R` (RST)<br>`U` (URG)<br>`P` (PSH)<br>`1` (Reserved bit 1)<br>`2` (Reserved bit 2)<br>`0` (No TCP flags set)<br>The first part defines the bits to match:<br>• The flags `S`, `A`, `F`, `R`, `U`, and `P` must be in upper case.<br>• If the first digit is `0`, it will stop and ignore all of the following flags.<br>• `*` matches any one of the specified bits.<br>• `+` matches all of the specified bits, plus any others.<br>• `!` matches if none of the specified bits is set.<br>• `Default` matches the specified bits exactly.<br>The second part is optional. It identifies the bits that should be masked off before matching.<br>Examples:<br>`--tcp_flags 0,12;`<br>`--tcp_flags !SAFRUP,12;`<br>`--tcp_flags S,12;`<br>`--tcp_flags S+;`<br>`--tcp_flags *SAFRUP12;` |
| `--window_size` | `[!]<number>;`<br>`[!]0x<number>;` | Check for the specified TCP window size.<br>• `!` = not equal.<br>• `0x` or `0X` = hexadecimal format<br>• `>` = greater than<br>• `<` = lesser than<br>Examples:<br>`--window_size 1000;`<br>`--window_size !0x1000;` |

**Table 3:** Check TCP header properties (continued)

| Keyword | Value | Description |
|---|---|---|
| `--tcp.src_port`<br>`--tcp.dst_port`<br>`--tcp.seq`<br>`--tcp.ack`<br>`--tcp.flags`<br>`--tcp.window_size`<br>`--tcp.checksum`<br>`--tcp.urgent`<br>`--tcp.any_option` | *operator value* | Check fields `src_port`, `dst_port`, `seq`, `ack`, `flags`, `window_size`, `checksum`, `urgent`, and `options` in TCP header. The syntax is:<br>`--tcp.[Decorations] operator value;`<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Examples:<br>`--tcp.src_port in [1111,2222];`<br>`--tcp.flags & 0x0f = 0x6`<br>`--tcp.any_option = 0x6052, dword;`<br>`//iterate over all options` |
| `--tcp[offset]` | *operator value* | Access any fields in TCP header in a free-lance mode. The syntax is:<br>`--tcp[offset] operator value [, word size] [, endianness]`<br>Both word size and endianness are optional. By default, the engine uses big endian and BYTE respectively.<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Examples:<br>`--tcp[20] &0xF0 = 0x30` |

## UDP header options

**Figure 4:** UDP packet format




The following options are available to check the UDP header:

**Table 4:** Check UDP header options

| Keyword | Value | Description |
|---|---|---|
| `--src_port` | `[!]<number>;`<br>`[!]:<number>;`<br>`[!]<number>:;`<br>`[!]<number>:<number>;` | Check the source port number or range.<br>• The prefix `!` means exclude.<br>• `:` is used to indicate a range.<br>Example:<br>`--src_port1000:;` |
| `--dst_port` | `[!]<number>;`<br>`[!]:<number>;`<br>`[!]<number>:;`<br>`[!]<number>:<number>;` | Check the destination port number or range.<br>• The prefix `!` means exclude.<br>• `:` is used to indicate a range.<br>Example:<br>`--dst_port 200:300;` |
| `--udp.src_port`<br>`--udp.dst_port`<br>`--udp.length`<br>`--udp.checksum` | *operator value* | Check fields `src_port`, `dst_port`, `length`, and `checksum` in the UDP header. The syntax is:<br>`--udp.[Decorations] operator value;`<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--udp.src_port in [1111,2222];` |
| `--udp[offset]` | *operator value* | Access any fields in TCP header in a free-lance mode. The syntax is:<br>`--tcp[offset] operator value [, word size] [, endianness]`<br>Both word size and endianness are optional. By default, the engine uses big endian and BYTE respectively.<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--udp[20] &0xF0 = 0x30` |

## ICMP header options

The details of the ICMPv4 packet format are shown in Figure 5. The ICMPv6 packet format is shown in Figure 6.

**Figure 5:** ICMPv4 packet format

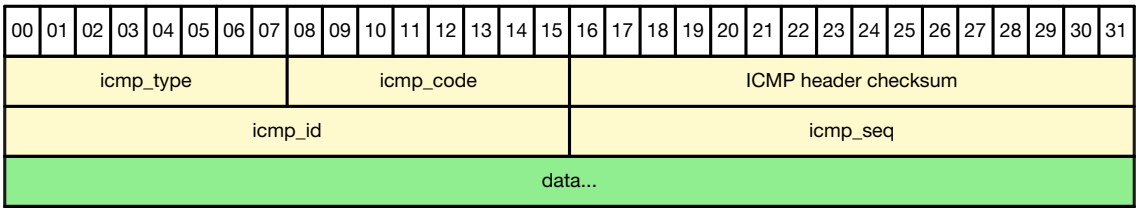



**Figure 6:** ICMPv6 packet format

| Bit offset | 0-7 | 8-15 | 16-31 |
|---|---|---|---|
| 0 | Type | Code | Checksum |
| 32 | Message body | | |

The following test are available to check the ICMP header:

**Table 5:** ICMP header tests

| Keyword | Value | Description |
|---|---|---|
| `--icmp_type` | `<number>;` | Specify the ICMP type to match. Cover both ICMPv4 and ICMPv6. |
| `--icmp_code` | `<number>;` | Specify the ICMP code to match. Cover both ICMPv4 and ICMPv6. |
| `--icmp_id` | `<number>;` | Check for the specified ICMP ID value.<br>This keyword is only used for packets with ICMP type `ECHO_REQUEST` or `ECHO_REPLY`. |
| `--icmp_seq` | `<number>;` | Check for the specified ICMP sequence value.<br>This keyword is only used for packets with ICMP type `ECHO_REQUEST` or `ECHO_REPLY`. |
| `--icmp.code`<br>`--icmp.type`<br>`--icmp.checksum` | *operator value* | Check fields `code`, `type`, and `checksum` in ICMPv4 header. The syntax is:<br>`--icmp.[Decorations] operator value;`<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--icmp.code in [1,2];` |

**Table 5:** ICMP header tests (continued)

| Keyword | Value | Description |
|---|---|---|
| `--icmp[offset]` | *operator value* | Access any fields in ICMPv4 header in a free-lance mode. The syntax is:<br>`--icmp[offset] operator value [, word size] [, endianness]`<br>Both word size and endianness are optional. By default, the engine uses big endian and BYTE respectively.<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--icmp[1] in [1,2];` |
| `--icmp6.code`<br>`--icmp6.type`<br>`--icmp6.checksum` | *operator value* | Check fields code, type, checksum in ICMPv6 header. The syntax is:<br>`--icmp6.[Decorations] operator value;`<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--icmp6.code = 135;` |
| `--icmp6[offset]` | *operator value* | Access any fields in ICMPv6 header in a free-lance mode. The syntax is:<br>`--icmp6[offset] operator value [, word size] [,endianness]`<br>Both word size and endianness are optional. By default, the engine uses big endian and BYTE respectively.<br>Operators you can use include: `=`, `!=`, `>=`, `<=`, `&`, `\|`, `^`, and `in`.<br>Example:<br>`--icmp6[0] = 135;` |

# Payload Related Options

IPS signatures use pattern matching for inspecting of a packet payload. A pattern definition starts with a `--pattern` or a `--pcre` option name, and is followed by a series of modifiers. The general format of a pattern definition is:

```
--pattern <string>; [--context c;] [--no_case;]
     [--distance n[,<refer>]]; [--within n[,<refer>]];
```

Or, for PCRE patterns:

```
--pcre <string>; [--context c;] [--distance n[,<refer>]];
     [--within n[,<refer>]];
```

## pattern

The `pattern` keyword is used to specify which content to match. The pattern can contain mixed text and binary data. The binary data is generally enclosed within the pipe "`|`" characters, and is represented as hexadecimal numbers. It can match content in all packets for all protocols.

If there is no `no_case` keyword following the `pattern` keyword, the content matching is case sensitive.

If there is no `distance` or `within` keyword following the `pattern` keyword, the content is searched for from the beginning of the packet to the end.

If `--context packet_origin` does not follow the `pattern` keyword, for decoded traffic like Telnet and BO2K, the content is searched for in the decoded buffer. So in order to search the original buffer, `--context packet_origin` should be used as a modifier for the `pattern` keyword.

The pattern to be matched must be enclosed in double quotation marks and followed by a semicolon. The special characters (" `;` `\` `|`) must be written as (`|22|`, `|3B|` or `|3b|`, `|5C|` or `|5c|`, `|7C|` or `|7c|`). Although a backslash ( `\` ) can be used to escape any character except "`;`", this is not recommended.

### Format

```
--pattern [!]"<text>"; //[!] indicates the content is matched if it
     does not appear in the packet.
```

### Examples

```
--pattern "/level/";
--pattern "|E8 D9FF FFFF|/bin/sh";
--pattern !"|20|RTSP/";
```

## pcre

The `pcre` keyword is used to specify the content to match using Perl Compatible Regular Expression (PCRE). For the PCRE syntax, please refer to http://perldoc.perl.org/perlre.html.

The pattern to be matched must be enclosed in double quotation marks and followed by a semicolon. The special characters (`" ; /`) must be written as (`\x22`, `\x3B` or `\x3b`, `\x2F` or `\x2f`).

As IPS Engine handles PCRE a lot slower compared to normal pattern matching. PCRE should be used very carefully, especially for signatures that detect traffic from HTTP servers or traffic that does not specify a port.

**Format**

```
--pcre [!]"/<regular expression>/[<op>]"; //[!] indicates the content
     is matched if it does not appear.
```

| <op> | Description |
|---|---|
| `i` | Case insensitive. |
| `s` | Include new lines in the dot ( `.` ) meta character. |
| `m` | By default, the string is treated as one big line of characters. `^` and `$` match at the beginning and ending of the string.<br>When `m` is set, `^` and `$` match immediately following or immediately before any new line in the buffer, as well as the very start and very end of the buffer. |
| `x` | White space data characters in the pattern are ignored except when escaped or inside a character class. |
| `A` | The pattern must match only at the start of the buffer (same as `^`). |
| `E` | Set `$` to match only at the end of the subject string. Without `E`, `$` also matches immediately before the final character if it is a newline, but not before any other newlines. |
| `G` | Inverts the *greediness* of the quantifiers so that they are not greedy by default, but become greedy if followed by `"?"`. |

**Example**

```
--pcre "/\sLIST\s[^\n]*?\s\{/smi";
```

## context

The `context` keyword is used to specify which protocol field the engine should search for a pattern in. If it is not present, IPS searches for the pattern in the whole packet. Refer to "Pattern context" on page 52 for a description of the fields in different protocols.

The `context` keyword accepts one of the following values:

| | |
|---|---|
| **PACKET** | Searches for the pattern in the whole packet. This is the default setting. |
| **PACKET_ORIGIN** | Searches the original packet without protocol decoding. |
| **URI** | This is only used to match content in the URI field of a HTTP request.<br>Since there are various encoding standards that can be used in a URI, a character can be expressed in several ways. For example `%2f`, `%u002f`, and `%c0%af` all represent "`/`". In order to cope with evasion attempts based on this, the content to be searched for in a URI must be decoded.<br>The HTTP dissector decodes and normalizes the original URI field, placing the results in three buffers. Here is a look at the purpose and contents of the buffers:<br>Original URI:<br>`/scripts/..%c0%af../winnt/system32/cmd.exe?/c+ver`<br>Decoded URI:<br>`/scripts/../../winnt/system32/cmd.exe?/c+ver`<br>("`\`" is also converted to "`/`" in this phase.)<br>Extracted Directory Traversal URI:<br>`/winnt/system32/cmd.exe?/c+ver`<br>All three URI buffers are searched for the specified pattern. |
| **HEADER** | The search range is the entire header section of a scanned HTTP, IMAP, SMTP, POP3 or SSH traffic. |
| **BODY** | The search range is the entire body section of a scanned HTTP, IMAP, SMTP, or POP3 traffic. The decoder has no separate buffer for the body section of above mentioned traffics. Because of this, body data in different packets are not reassembled. The decoder just locates the beginning and end of the body in a packet payload and tries to match inside of it. Hence, If a signature has two patterns in a body section that are to be matched, but the patterns span across two separate packets, the second pattern will not be matched. |
| **BANNER** | The search range is the banner section of a scanned HTTP, IMAP. SMTP, POP3, or SSH traffic. |

| | |
|---|---|
| **HOST** | For HTTP session, The search range is the "`Host:`" field of a HTTP header. Refer to "HTTP/SIP context" on page 52 for more details.<br><br>For HTTPS session, The search range is the server name field of SNI (Server Name Extention) in client Hello packet and the CN (Common Name) field of certificate in the server Certificate packet. Refer to "SSL context" on page 55 for more details. |
| **FILE** | The search range for the file context can be one of:<br>• Decoded attachments for email protocols.<br>• Data sessions for FTP.<br>• The body for HTTP. |

**Notes**

IPS engine supports so called "packet-based" inspection. It will inspect packets even if there are no sessions associated with them. For example a TCP packet without threeway hand shaking. Many keywords, for example those for matching TCP/IP header fields, are enabled in "packet-based" inspection. If a pattern has the context value "`PACKET`", "`PACKET_ORIGIN`", or no context, it will be inspected using "packet-based" inspection.

The BANNER and BODY are in the packet buffer.

There is no body context in FTP, so file context should be used instead.

For HTTP, the body context and the file context are the same. You can use either `--context file` or `--context body` to indicate where to match the pattern.

MIME parsing is supported for the email protocols SMTP, IMAP, POP3 and NNTP. We decode most of the encoding methods, including base64, uuencode, 7/8bit, quota, binary and quoted-printable.

If the file itself is zipped or archived, the engine currently does NOT decompress it.

**Format**

```
--context <field>;
```

**Examples**

```
--context URI;
--context PACKET_ORIGIN;
```

## no_case

The `no_case` keyword is used to indicate that the pattern should be matched in a case insensitive manner.

**Format**

```
--no_case;
```

**Examples**

```
--no_case;
```

## distance, distance_abs, within, within_abs

These four keywords are used to specify the range (in bytes) of where the engine will search for a pattern. The keyword `distance` indicates the offset from the last reference point to start searching for a pattern, while `within` indicates the range of bytes from the last reference point which the engine should search for a pattern.

**Format**

```
--distance <range>[,<refer>];
--distance_abs <range>[,<refer>];
--within <range>[,<refer>];
--within_abs <range>[,<refer>];
```

`<range>` is a positive or negative numeric value.

The `<refer>` field is the reference point for the `<range>`. If it is not included the default is `MATCH`.

| | |
|---|---|
| **MATCH** | The reference is the last matched pattern. This is the default setting. |
| **PACKET** | The reference is the beginning of the packet. |
| **CONTEXT** | The reference is the beginning of the pattern context. |
| **REVERSE** | Search for the pattern relative to the end of the packet or context. This is only accepted with the `--distance` option, and the reference must be packet or context. |
| **LASTTAG** | The reference is the one set by last PSET. Refer to Figure 17 on page 58 for more details. |

**Notes**

If the keywords `distance` and `within` are used with the first pattern of a signature, the `<refer>` field should be set to `context`, as there are no previous matched patterns.

The keywords `distance` and `distance_abs` indicate the minimum distance from the end of the last reference point to the beginning of the current pattern. The distance is counted from the next character after the last reference point. Both `distance` and `distance_abs` support negative range value. In this case, `distance` does not require enough data before reference point while `distance_abs` does. For example, the following signature makes sure no "`?`" character is before the "`/BBBB`" pattern in the URI:

```
---pattern "/BBBB"; --context uri; --within 200,context; --pattern
     !"?"; --context uri; --distance 200; --within 200;
```

It works even if there are no 200 bytes of data before the "`/BBBB`" pattern in the URI.

The keywords `within` and `within_abs` indicate that the whole pattern must appear within the given range following the last reference point. If the `distance` or `distance_abs` keywords, with the same reference point, are also present, the pattern will be matched from the specified distance to the range of bytes specified in the `within` or `within_abs` keywords.

The keywords `distance_abs` and `within_abs` should be used only for negative matches (patterns with the "`!`" modifier). They indicate that the buffer following the reference point must be longer than or equal to the value specified by `<range>`. Compare the following two cases:

```
--pattern !"|0a|"; --within 100,match;
--pattern !"|0a|"; --within_abs 100,match;
```

If the buffer after the previous match is shorter than 100, the first signature is matched. Although the latest IPS engine does not forbid the usage of `distance_abs` and `within_abs` for positive match, it is not recommended to do so. The behavior of the keyword `within_abs` for a positive match is tricky. Therefore, It is better to use the keyword `data_at` instead. For example:

Not Recommended:

```
--pattern “BBBBBB”; --pattern “DDDDDD”; --within_abs 200;
```

Recommended:

```
--pattern “BBBBBB”; --data_at 200,relative; --pattern “DDDDDD”;
    --within 200;
```

These two signatures are equivalent while the latter is more readable. A negative `<range>` value can be used to specify the range before the reference. Different types and references can be combined as range modifiers. See “Distance within diagram” on page 57 for examples.

**Examples**

```
--pattern "/disp_album.php?"; --context uri; --no_case;
    --within 50,context;
--pattern "|05 00|"; --distance 0; --pattern "|6e 00|"; --distance 5;
    --within 2;
--pattern "Host: "; --context header; --pattern !"|0a|";
    --context header; --within_abs 80; --distance 10,packet,reverse;
    --within 5,packet; //(First count 10 bytes back from the end of
    the packet, then search for the pattern within 5 bytes.)
```

## byte_jump, byte_test

The `byte_jump` keyword is used to move the reference point. The distance to be skipped is calculated from the value of bytes at a specified offset. For more details see Figure 16 on page 58.

The `byte_test` keyword is used to compare the value of bytes at the specified offset with a given value. It does not move the reference point.

If the data to be processed or skipped is beyond the end of the packet, the option is considered unmatched.

`byte_test` and `byte_jump` cannot be used in the URI field.

**Format**

```
--byte_jump <*|bytes>,<offset>[,<multiplier>[,<modifiers>]];
--byte_test <*|bytes>,<op>,<value>,<offset>[,<multiplier>[,<modifier
     s>]];
```

| | |
|---|---|
| `<*\|bytes>` | Specifies the number of bytes from the payload to be converted. The value to be converted can be an ASCII string or binary.<br>If the value is in binary, select between 1, 2 or 4 bytes to be converted.<br>If the value is an ASCII string, use the "`string`" modifier. For a fixed length ASCII field specify the field's length. If it is a variable length ASCII field, use "`*`", which will convert all bytes from the offset until the first nondigit character in the chosen base has been detected. |
| `<op>` | Defines the operator used to compare the value converted from the packet with the value specified. The following operators are accepted:<br>`>` The value converted must be larger than the value specified.<br>`<` The value converted must be smaller than the value specified.<br>`=` The value converted must be equal to the value specified.<br>`!` The value converted must be not equal to the value specified.<br>`&` The value converted AND the value specified must be not equal to zero.<br>`~` The value converted AND the value specified must be equal to zero.<br>`^` The value converted XOR the value specified must be not equal to zero. |
| `<value>` | Specifies the value to be compared with. A hexadecimal number can be specified with the prefix 0x.<br>This also accepts variables and arithmetic operations (`+ * /`).<br>The following predefined variable is accepted:<br>• `$PKT_SIZE`: The data will be compared with the packet size. |
| `<offset>` | Specifies the starting point where content should be converted in the payload. Negative offsets are accepted. See the modifier `relative` (below) for more details. |

| | |
|---|---|
| <multiplier> | Optional. It must be a numerical value when present. The converted value multiplied by this number is the result to be compared or skipped. |
| <modifiers> | Accepts a combination (separated by ",") of the following values:<br>`relative` Indicates that the offset should start from the last match point. Without it, the offset starts from the beginning of the packet.<br>`big` Indicates that the data to be converted is in big endian (default).<br>`little` Indicates that the data to be converted is in little endian.<br>`string` Indicates that the data to be converted is a string.<br>`hex` Indicates that the string data to be converted is in hexadecimal.<br>`dec` Indicates that the string data to be converted is in decimal (default).<br>`oct` Indicates that the string data to be converted is in octal.<br>`align` Rounds the number of converted bytes up to the next 32bit boundary, only used with byte_jump. |

**Examples**

```
--byte_jump 4,0,relative;
--byte_test 4,>,3536,0,relative;
--byte_jump 4,20,relative,align;
--byte_jump 4,0,4,relative,little;
--byte_test 4,>,0x7FFF,4,relative;
--byte_test 4,>,$PKT_SIZE,4,relative;
--byte_test 4,>,$PKT_SIZE4,2,relative;
```

# Special Options

## service

The `service` keyword is used to specify the session type associated with a packet. The session type is identified by dissectors. So in order for this keyword to work, the session that is being identify should be supported by a suitable dissector. Refer to “Service option types” on page 48 for details about services that are currently support by the IPS Engine dissectors.

A signature must contain "`--service <service name>;`" to detect packets that belong to a service that the IPS Engine supports. For details see “IPS engine service logic” on page 49.

The service value depends entirely on the IPS Engine’s ability to identify the session type.

**Format**

```
--service <service name>;
```

**Examples**

```
--service HTTP;
--service DNS;
```

## flow

The `flow` keyword is used to specify the direction of the detection packet. It can be applied to TCP and UDP sessions. It accepts one of the following direction values:

| | |
|---|---|
| **FROM_CLIENT** | Match packets sent from the client to the server. |
| **FROM_SERVER** | Match packets sent from the server to the client. |
| **BI_DIRECTION** | Match packets from client to server and from server to client. |
| **REVERSED** | Specifies that the attack is in the opposite direction from the detected packet. A typical case is when a bruteforce login is detected by matching a server packet indicating that a login has failed. This keyword will not affect detection. Its purpose is to tell the GUI to display the correct location for the vulnerability (Client or Server). |

**Format**

```
--flow <direction>;
```

**Examples**

```
--flow from_client;
--flow bi_direction; dst_port 123; // match if source or destination
     port is 123.
```

## tag

The `tag` keyword is used in a signature to mark a session with a named tag, or to check whether a tag has been set for a session.

Pattern matching with IPS signatures is essentially packet based. The tag keyword is mainly used when attack patterns appear in more than one packet or in different directions. A signature that matches an earlier packet in an attack can mark the session with a named tag, and the existence of the tag can be tested when ensuing packets in the same session are scanned.

The matching algorithm guarantees the order in which rules are scanned. The rules are sorted based on their tag dependencies. During packet inspection, the rules are matched in this order, so that signatures that depend on other signatures are always scanned later in the process.

The `<op>` value determines which operation is performed.

The name of a tag should only contain printable characters. It should not contain spaces, commas, '`!`', or semicolons.

By default, a newly created tag is in the un-set state.

Patterns in `tag set` and `tag test` rules can appear in the same packet together.

### Format

```
--tag <op>,[!]<name>;
```

`<name>` Indicates the name of a tag.

`[!]` Only allowed in TEST operations. It returns true if the tag does not exist.

`<op>` Accepts one of the following values:

| | |
|---|---|
| **SET** | Mark the session with a named tag. |
| **PSET** | Mark the session with a named tag and remember the last reference point. This reference point can be referred by using LASTTAG for keywords `distance`/`within`/`distance_abs`/`within_abs`. Refer to Figure 17 on page 58 for more details. |
| **CLEAR** | Remove the specified tag from the session. |
| **TOGGLE** | Toggle the specified tag (SET<=>CLEAR) in the session. |
| **TEST** | Test the existence of the specified tag. Add "`!`" if the signature is to test nonexistence of the tag. |
| **RESET** | Clear ALL tags from the session. |
| **QUIET** | Suppress logging when the signature is matched, and cause the signature's action to be ignored. QUIET is normally included in the signature that SET the tag. Signatures with `--tag set;` should also have `--tag quiet` and `--status hidden;`. |

### Examples

```
--tag quiet; tag set,Tag.Rsync.Argument;
--tag quiet; tag clear,tag.login;
--tag test,Tag.Rsync.Argument;
--tag test,!DHTML.EDIT.CONTROL.CLSID;
```

## parsed_type

The `parsed_type` keyword is used to match a packet or session attribute that can be identified by the dissectors. A signature can have more than one `--parsed_ type` keyword.

**Format**

`--parsed_type <type>;`

`<type>` Can take any one of the following values:

| | |
|---|---|
| **SSL_PCT**<br>**SSL_V2**<br>**SSL_V3**<br>**TLS_V1**<br>**TLS_V2** | These types are used to identify the SSL and TLS versions. |
| **SOCK4**<br>**SOCK5** | These match sessions using the SOCKS 4 or SOCKS 5 proxy protocols. |
| **HTTP_GET** | The HTTP request method to be matched is GET. This is valid for the lifetime of the request.<br><br>In most cases, a signature using `--parsed_type`, similar to the one below:<br><br>`--service HTTP; --parsed_type HTTP_GET;`<br><br>Can replace a pattern based one like this:<br><br>`--service HTTP; --pattern "GET\|20\|" context uri; --within 4,context;`<br><br>But sometimes the "GET" string must be checked explicitly:<br><br>• If it is not an HTTP session.<br>• When additional bytes must be matched, for example:<br>`--service HTTP; --pattern "GET\|20\|"; context uri; --within 4,context; --pattern "\|13 12 13\|"; --context uri; --distance 0;` |
| **HTTP_POST** | The HTTP request method to be matched is POST. This is valid for the lifetime of the request. |
| **HTTP_CHUNKED** | The Transfer-Encoding type of the HTTP request to be matched is chunked. This is valid for the lifetime of the request.<br><br>In most cases a signature using the `parsed_type` keyword, similar to the one below:<br><br>`--parsed_type HTTP_CHUNKED;`<br><br>Can replace one that looks for strings, like this:<br><br>`--service HTTP; --pattern "TransferEncoding"; --context header; --no_case; --pattern "chunked"; --context header; --no_case; --distance 1;` |

### Examples

The following are two versions of the same HTTP signature, the second one is faster and more accurate.

**1.**

```
F-SBID( --name "FrontPage.Fp30reg.Chunked.Overflow"; --protocol tcp;
     --service HTTP; --flow from_client; --pattern "POST";
     --context uri; --distance 0,context; --within 5,context;
     --pattern "/_vti_bin/_vti_aut/fp30reg.dll"; --context uri;
     --no_case; --distance 0; --pattern "TransferEncoding";
     --context header; --no_case; --pattern "chunked";
     --context header; no_case; --distance 1; )
```

**2.**

```
F-SBID( --name "FrontPage.Fp30reg.Chunked.Overflow"; --protocol tcp;
     --service HTTP; --flow from_client; --parsed_type HTTP_POST;
     --pattern "/_vti_bin/_vti_aut/fp30reg.dll"; --context uri;
     --no_case; --parsed_type HTTP_CHUNKED; )
```

These are two SSL signatures that detect the same vulnerability. The second one is better than the first one.

**1.**

```
F-SBID( -name "SSL.PCT.Overflow"; --protocol tcp; --dst_port 443;
     --flow from_client; --tag test,!Tag.SSLv3.Web.443;
     --tag test,!Tag.SSLv2.Web.443; --tag test,!Tag.TLSv1.Web.443;
     --pattern "|01|"; --within 1,packet; --distance 2,packet;
     --byte_test 2,>,1,3; --byte_test 2,<,0x301,3; --byte_test
     2,>,0,5; --byte_test 2,!,0,7; --byte_test 2,<,16,7; --byte_test
     2,>,16,9; --byte_test 2,<,33,9; --pattern "|8F|";
     --within 1,packet; --distance 11,packet;
     --byte_test 2,>,32768,0,relative; --data_size >300; )
```

**2.**

```
F-SBID( --name "SSL.PCT.Overflow"; --protocol tcp;
     --flow from_client; --service SSL; --parsed_type SSL_PCT;
     --pattern "|01|"; --within 1,packet; --distance 2,packet;
     --byte_test 2,>,1,3; --byte_test 2,<,0x301,3; --byte_test
     2,>,0,5; --byte_test 2,!,0,7; --byte_test 2,<,16,7; --byte_test
     2,>,16,9; --byte_test 2,<,33,9; --pattern "|8F|";
     --within 1,packet; --distance 11,packet;
     --byte_test 2,>,32768,0,relative; --data_size >300; )
```

## dhcp_type

The `dhcp_type` keyword is used to match DHCP request/response types. Any numeric value is allowed. The following table shows the types defined in RFC.

**Table 6:** Types defined in RFC

| | |
|---|---|
| DISCOVER | 1 |
| OFFER | 2 |
| REQUEST | 3 |
| DECLINE | 4 |
| ACK | 5 |
| NAK | 6 |
| RELEASE | 7 |
| INFORM | 8 |
| RELAY_CLIENTREQUEST | 9 |
| RELAY_SERVERREPLY | 10 |

**Format:**

```
--dhcp_type <value>;
```

**Example**

```
--dhcp_type 1;
```

## data_size

The `data_size` keyword was originally used to test the TCP/UDP/ICMP payload size of the packet being inspected. It has since been extended to support other size related fields in application protocols.

Because TCP is stream based, not packet based, the sender can intentionally fragment the original packets before they are transmitted to evade detection. For this reason using data_size on TCP packets may not always be reliable.

### Format

```
--data_size [op]<value>[,field];
```

`[op]` is not required. When it is not present, the default operator is "=". The following operators are accepted:

| | |
|---|---|
| **>** | The data size must be larger than the value specified. |
| **<** | The data size must be smaller than the value specified. |
| **=** | The data size must be equal to the value specified. |

`<value>` is required. It is a decimal number that specifies the data size.

`[field]` is optional. When it is not present, the packet PAYLOAD size is checked. For other actions, one of the following keywords can be used:

| | |
|---|---|
| **PAYLOAD** | The TCP/UDP/ICMP payload size is checked. This is the default setting. |
| **URI** | The URI length is checked. |
| **HEADER** | The length of the Header is checked. |
| **BODY** | The length of the Body is checked. |
| **HTTP_CONTENT** | The value of "`Content-Length:`" in a HTTP header is checked. |
| **HTTP_CHUNK** | The chunk length value in the chunk header is checked. |
| **HTTP_HOST** | The length of the "`HOST:`" line in an HTTP header is checked. The length count includes CRLF characters, the field name "`HOST:`", all white spaces between the field name to the field value, and the field value . e.g. "`HOST: www.example.com\r\n`" will have a data_size of 25. |
| **SMTP_BDAT** | The SMTP data length in a BDAT command is checked. |
| **SMTP_XEXCH50** | The SMTP data length in an XEXCH50 command is checked. |

### Examples

```
--data_size <128;
--pattern "/admin_/help/"; --context uri; --no_case;
     --data_size >1024,uri;
--parsed_type HTTP_POST; --pattern "nsiislog.dll"; --context uri;
     --no_case; --data_size >1000,http_content;
```

## rate and track

These two keywords make it possible to tell the IPS engine that instead of triggering a signature every time it is matched, it should only trigger if the signature is matched a given number of times within a specified time period. This feature can be used in reporting slow port scans, brute-force login attempts, and similar behavior.

For a regular signature, the IPS engine first compares all of the keyword options other than rate and track. If all the options are matched, IPS checks whether rate is specified for the signature. If it is not, IPS triggers the signature. If it is, IPS increases the counter and updates the timestamp, and checks whether the trigger rate has been reached.

**Format**

```
--rate<count>,<duration>[,limit];
```

| | |
|---|---|
| **<count>** | The number of matches that must be seen before a log entry is generated. |
| **<duration>** | The time period over which matches are counted, in seconds. |
| **[limit]** | This improves the accuracy of the matched packet count by counting in strict time rather than averaging over a period of time.<br>For example, "`rate 400,1,limit;`". |

```
--track <keyword>;
```

`<keyword>` specifies the packet property to track. The following case insensitive keywords are accepted:

| | |
|---|---|
| **SRC_IP** | Track the packet's source IP address. |
| **DST_IP** | Track the packet's destination IP address. |
| **DHCP_CLIENT** | Track the DHCP client's MAC address. |
| **DNS_DOMAIN** | Track the domain name in the DNS query record. |
| **DNS_DOMAIN_AND_IP** | Track the DNS response with same domain name and IP address. |

**Notes**

If `--track` is specified, only matched packets which have the same specified tracked are added to the counter.

If `--rate` is used without `--track`, all matched packets are added to the counter and the signature is reported once the threshold is reached.

IPS counts the average number of packets over a period of time. This might allow some extra packets to go through. Therefore, to have better accuracy, the `limit` keyword was added to allow counting to be done in a strict time. When limit is enabled the packet count is more accurate.

**Example**

```
F-SBID( --name DHCP.FLOOD; --protocol UDP; --service DHCP;
     --dhcp_type 1; --rate 100,10; --track DHCP_CLIENT; )
```

This signature indicates that if IPS sees DHCP discover requests (`--dhcp_type 1;`) more than 100 times within 10 seconds (`--rate 100,10;`) from the same DHCP client (`--track dhcp_client;`), then an alert is generated.

## log

The `log` option is used to specify additional types of information that can be included in the log messages generated by a signature.

**Format**

`--log <keyword>;`

`<keyword>` can be set to the following case insensitive values:

| | |
|---|---|
| **DHCP_CLIENT** | DHCP client MAC addresses will be added to the log message in the format "`dhcp_client=xx:xx:xx:xx:xx:xx;`". |
| **DHCP_CC_ID** | Circuit ID in DHCP relay messages will be added to the log in the format "`dhcp_cc_id=4F4C2D30303143;`". |
| **DNS_QUERY** | DNS query strings will be added to the log in the format "`dns_query=www.yahoo.com;`". |

**Example**

```
--log DHCP_CLIENT;
```

## data_at

The `data_at` keyword is used to verify that there is data at the specified location in the payload.

**Format**

`--data_at <number>[,relative];`

`<number>` is the payload offset to be checked for data.

`[relative]` indicates that the offset is relative to the end of the previous content match.

**Examples**

```
--data_at 100,relative;
```

## rpc_num

The `rpc_num` keyword is used to check the RPC application, version and procedure numbers in a SUNRPC CALL request.

**Format**

`--rpc_num <number1>[,<number2>[,<number3>]];`

| | |
|---|---|
| `<number1>` | The application number. |
| `[<number2>]` | The version number. |
| `[<number3>]` | The procedure number. |

`<number2>` and `<number3>` are optional.

"`*`" means that the number can have any value.

### Examples

```
--rpc_num 100221,*,*;
--rpc_num 100005,2,*;
--rpc_num 100000,*,4;
```

## file_type

The `file_type` keyword matches a class of file types, where each class contains several related subtypes. The IPS engine file type matching uses "file magic" to decide what type of file the content is, working in a manner similar to the linux "`file`" command.

Currently, for the HTTP protocol, the first 13 or more bytes of body content will be categorized into a file type. If the result is a subtype of the class specified by a `--file_type <class>` option in a signature, it is a match.

In most cases, the identification of file type is handled by the file type function. However, when you are unsure about the file type, you can rely on the protocol fields if they contain some fields such as "`content-type`". So file type may not be limited to the subtype listed below. For example, a tiff file will be marked as file type IMAGE by our IPS engine, even though it is not included in our own file type function.

The feature works in this way:

1. The traffic is parsed by the protocol decoder.
2. A check is done to determine the presence of a file, for HTTP, MIME and FTP.
3. If the decoder finds that there is a file in the traffic, it will call the file type function to identify what type of file it is.
4. To narrow down the file type results, a class is selected based on the file type.
5. The result is saved with the protocol, for signature use. If a signature includes this keyword it will check whether the given type has been matched.

### Format

```
--file_type <class>
```

### Notes

The file type classes are listed below along with their associated subtypes:

| | |
|---|---|
| **COMPRESS** | arj, bzip, bzip2, cab, gzip, lzh, lzw, rar, rpm, tar, upx, zip |
| **IMAGE** | gif, gif87a, gif89a, jpeg, png |
| **SCRIPT** | .bat, .css, .hta, .vba, .vbs, genscript, javascript, perlscript, shellscript, wordbasic |
| **VIDEO** | .avi, MPEG |
| **AUDIO** | .mp3 |
| **STREAM** | stream |
| **MSOFFICE** | MSOFFICE, PPT |
| **PDF** | .pdf |
| **FLASH** | FLASH |

| | |
|---|---|
| **EXE** | .com, .dll, .exe |
| **HTML** | HTML |
| **XML** | XML, WORDML |
| **UNKNOWN** | unknown, ActiveMIME, AIM, FORM, HLP, MIME, .txt |

**Examples**

```
--file_type PDF;
--file_type EXE;
```

## crc32

The keyword `crc32` is introduced to help in detection of file based vulnerability.

**Format**

`--crc32 <checksum>,<file_length>;`

`<checksum>` is required. It is a hexadecimal number that represent the crc32 checksum of the file.

`<file_length>` is required. It is a decimal number that represent the file length.

**Example:**

```
--crc32 3174B5C8,20480;
```

# Deprecated Options

These keywords are still supported in order to maintain backward compatibility:

```
uri
header
body
content
raw
offset
depth
```

# Appendix A: Engine and Option Details

## Service option types

Protocol dissectors are used to identify some well known services. The service option is used to match the session type, which is determined by the IPS Engine dissectors. Currently, the following service types are supported.

**Table 7:** Supported service types

| Session Type | Criterion | Service Option |
|---|---|---|
| Back_orifice (bo, bo2k) | TCP/UDP, any port | service BO |
| DCE RPC | TCP/UDP, anyport | service DCERPC |
| DHCP | UDP, any port | service DHCP |
| DNP3 | TCP, any port | service DNP3 |
| DNS | TCP/UDP, 53 | service DNS |
| FTP | TCP, any port | service FTP |
| H323 | TCP, 1720 | service H323 |
| HTTP | TCP, any port | service HTTP |
| IM (yahoo, msn, aim, qq) | TCP/UDP, any port | service IM |
| IMAP | TCP, any port | service IMAP |
| LDAP | TCP, 389 | service LDAP |
| MSSQL | TCP, 1433 | service MSSQL |
| NBSS | TCP, 139, 445 | service NBSS |
| NNTP | TCP, any port | service NNTP |
| P2P<br>(skype, BT, eDonkey, kazaz, gnutella, dc++) | TCP/UDP, any port | service P2P |
| POP3 | TCP, any port | service POP3 |
| RADIUS | UDP, 1812, 1813 | service RADIUS |
| RDT | TCP, any port, by RTSP | service RDT |
| RTCP | TCP, any port, by RTSP | service RTCP |
| RTP | TCP, any port, by RTSP | service RTP |
| RTSP | TCP, any port | service RTSP |
| SCCP (skinny) | TCP, 2000 | service SCCP |

**Table 7:** Supported service types (continued)

| Session Type | Criterion | Service Option |
|---|---|---|
| SIP | TCP/UDP, any port | service SIP |
| SMTP | TCP, any port | service SMTP |
| SNMP | UDP, 161, 162 | service SNMP |
| SSH | TCP, any port | service SSH |
| SSL | TCP, any port | service SSL |
| SUN | RPC TCP/UDP, 111, 32771 | service RPC |
| TELNET | TCP, 23 service | TELNET |
| TFN | ICMP, any port | service TFN |

## IPS engine service logic

In IPS Engine 3.0, if a signature has the `--service` keyword, it will be added to the corresponding service tree. A signature's service tree assignment determines which packets it will scan. IPS has multiple service trees (http, smtp, pop3, dns etc.) and one `unknown_service` tree.

If a signature has no `--service` keyword, but it has the `port` keyword, it will be added into the `unknown_service` tree.

If a signature has neither, it is a generic signature, and will be added to all service trees including the `unknown_service` tree.

If a packet is marked by a protocol dissector as some type of service, for example HTTP, it will only be inspected by signatures in the HTTP service tree. Because of this, if a signature uses "`--dst_port 80`" instead of "`--service HTTP`", it will not be matched. So you must ensure that all signatures detecting traffics with an implemented service type have the `--service` keyword.

However, since you cannot force external users to follow this rule, custom signatures where the type is `unknown_service` are added to all service trees.

Another important reason for using the `--service` keyword is to scan traffic not running on the standard service port. This is why our IPS engine tries to mark traffic based on packet content instead of portmapping. For example, although the default port for HTTP is 80, it can be modified based on configuration and systems. However, our IPS can still recognize the type of the service based simply on a packet’s content and mark the packet as HTTP.

## Signature options

All signatures and groups must have certain basic options defined. Pattern signatures require additional options to describe the properties of a packet. The tables in this section provide more information about the options and their uses.

In the tables the option subscripts indicate these requirements:

| | | |
|---|---|---|
| `Option` $_{m}$ | `m` | Required in the indicated signature type. |
| `Option` $_{s}$ | `s` | Single use, can only appear once in a signature. |

**Table 8:** Basic options

| Options | Pattern Signature Group | Pattern Signature | Dissector Group | Dissector Signature | Anomaly Signature | Custom Signature |
|---|---|---|---|---|---|---|
| `--name` $_{m,s}$ | X | X | X | X | X | X |
| `--service` $_{s}$ | | X | | X | | |
| `--flow` $_{s}$ | | X | | X | | |

**Table 9:** Pattern signature options

| Protocol Options | Payload Options | Special Options | Application Options |
|---|---|---|---|
| `--protocol` | `--pattern` | `--service` | `--app_cat` |
| `--ip_id` | `--pcre` | `--flow` | `--app_cat_sub` |
| `--ip_tos` | `--no_case` | `--tag` | `--pop` |
| `--ip_ttl` | `--context` | `--parsed_type` | `--risk` |
| `--ip_option` | `--distance` | `--dhcp_type` | `--technology` |
| `--same_ip` | `--distance_abs` | `--rate` | `--behavior` |
| `--src_addr` | `--within` | `--track` | `--vendor` |
| `--dst_addr` | `--within_abs` | `--data_--size` | |
| `--dst_port` | `--byte_jump` | `--crc32` | |
| `--src_port` | `--byte_test` | `--file_type` | |
| `--tcp_flags` | | `--data_at` | |
| `--seq s` | | `--log` | |
| `--ack` | | `--rpc_num` | |
| `--window_size` | | | |

**Table 9:** Pattern signature options (continued)

| Protocol Options | Payload Options | Special Options | Application Options |
|---|---|---|---|
| `--icmp_type` | | | |
| `--icmp_id` | | | |
| `--icmp_code` | | | |
| `--icmp_seq` | | | |

## How IPS triggers an alert

There are dissector signatures and pattern signatures. The results for dissector signature are generated by the protocol dissectors, while the results for pattern signature are generated by pattern matching.

1. A packet is first checked by the dissectors. Depending on the implementation of the dissectors, one packet could trigger multiple alerts. All of them are reported.
2. If the action for the dissector signatures in Step 1 is prevention, the packet is blocked and pattern matching will be skipped. Otherwise, if the action is PASS, the packet is inspected by the pattern signatures. If multiple signatures are matched, the best one is picked, based on the following criteria, in order:
   - The signature is not hidden.
   - The signature takes preventive action.
   - The signature has a higher severity level.
   - The signature has the longest single pattern.
3. If an alert is triggered in the same way, only the first alert will be reported. An example would occur in the case of TCP retransmission or fragmentation. The same action will be applied to retransmitted packets.
4. If multiple identical alerts are triggered, they will be aggregated. That is, if the same alert is triggered multiple times within a short period of time, the alerts will be aggregated into a single alert.

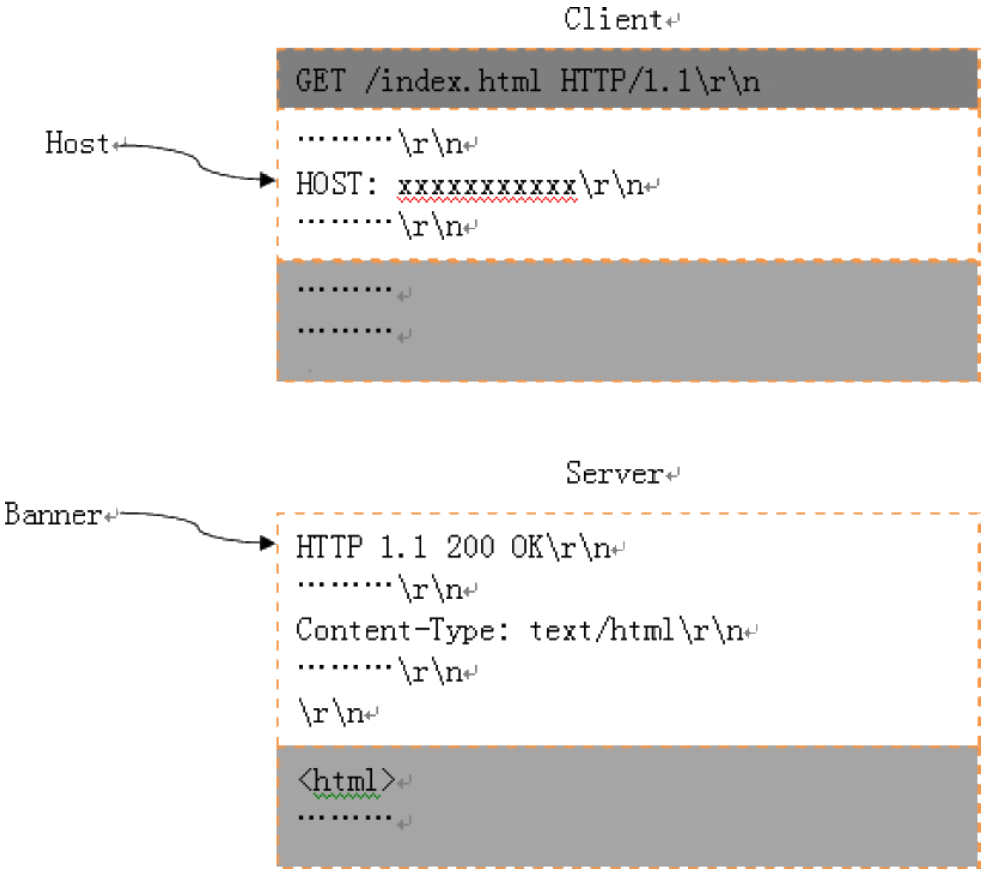
# Appendix B: Option Diagrams

## Pattern context

**Figure 7:** Legend

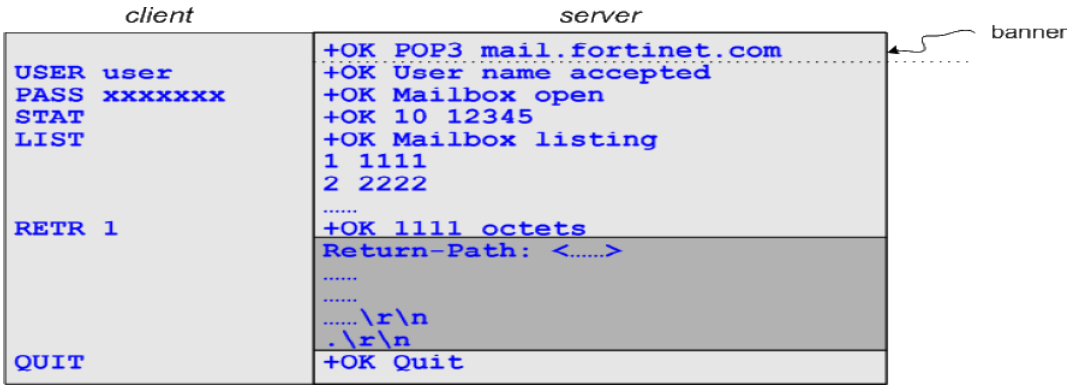

### HTTP/SIP context

**Figure 8:** HTTP/SIP context



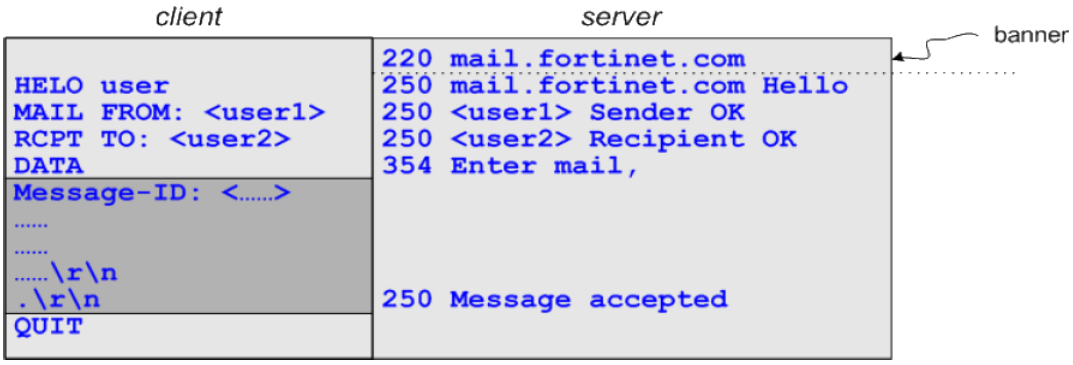
## POP3 context

**Figure 9:** POP3 context



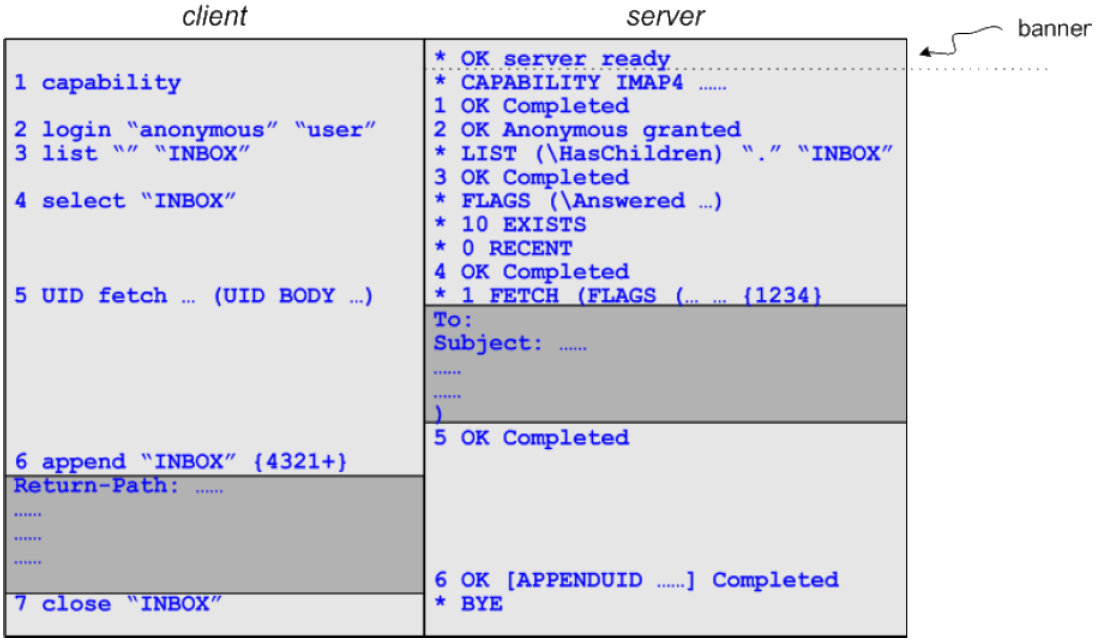

## SMTP context

**Figure 10:**SMTP context



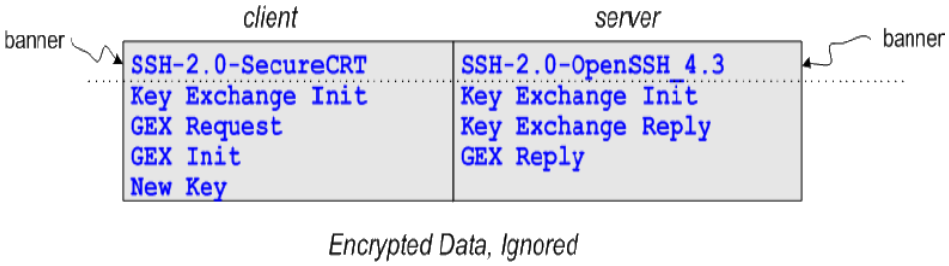
## IMAP context

**Figure 11:**IMAP context




## SSH context

**Figure 12:**SSH context



## SSL context

**Figure 13:**SSL context

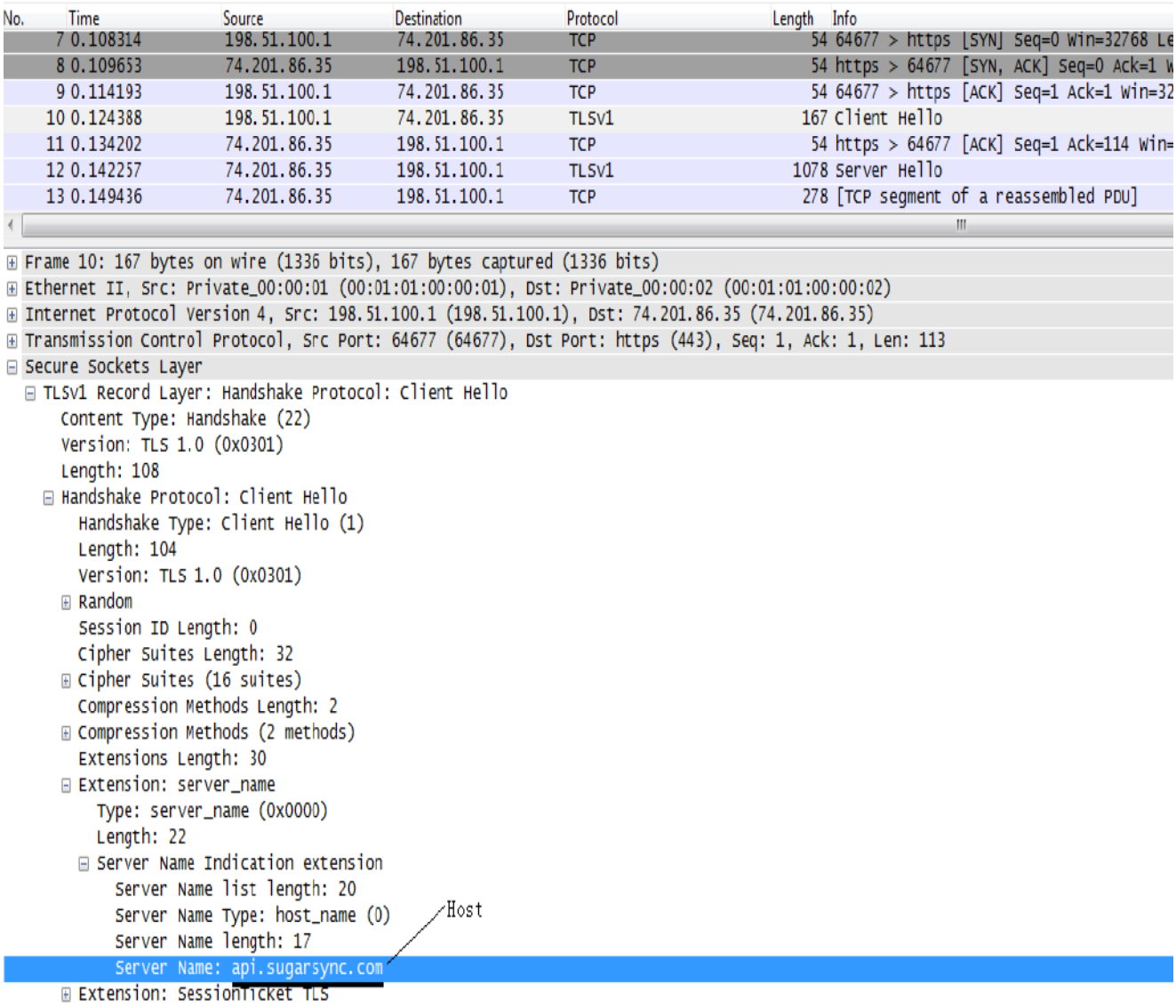
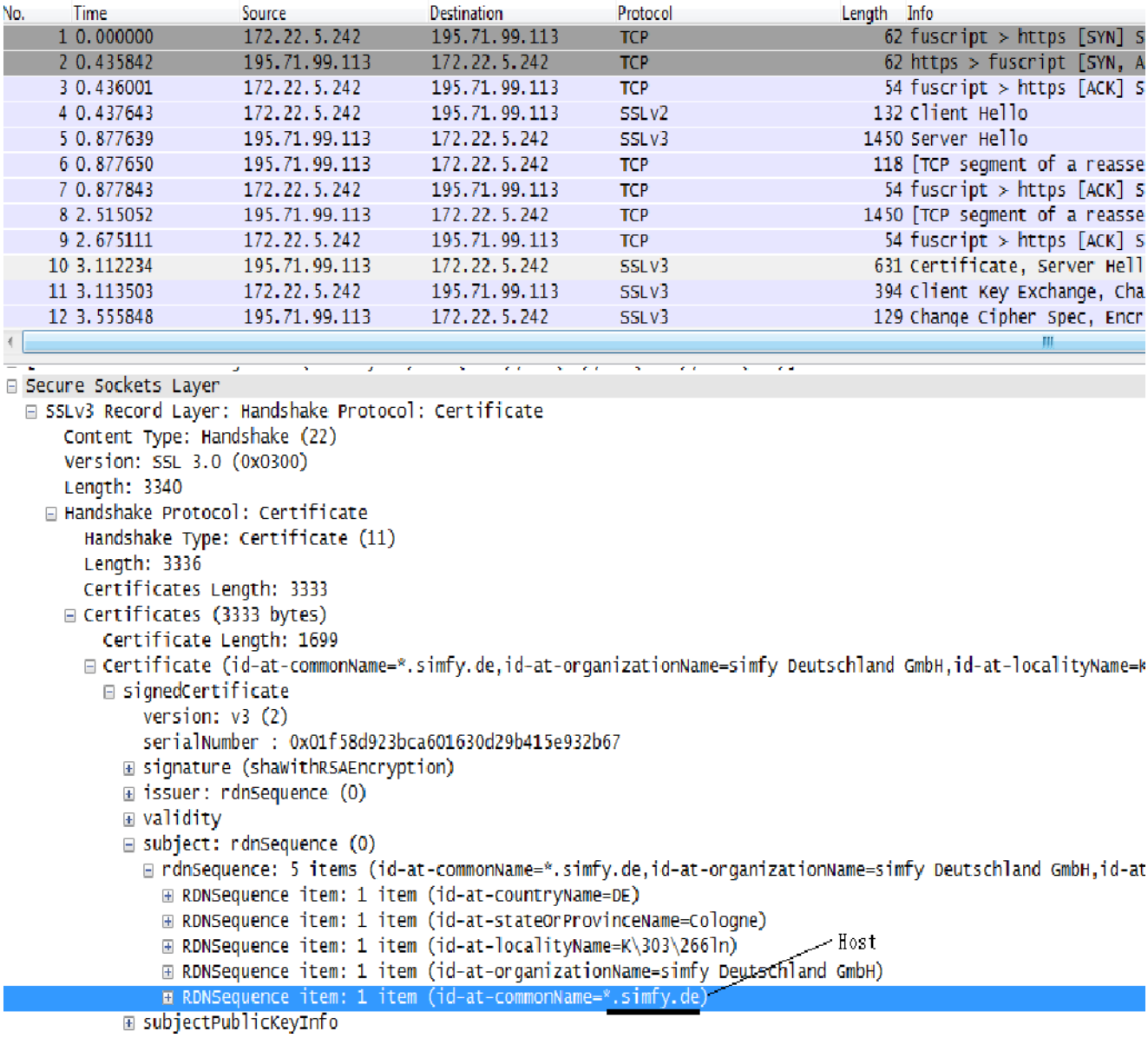


**Figure 14:**SSL context

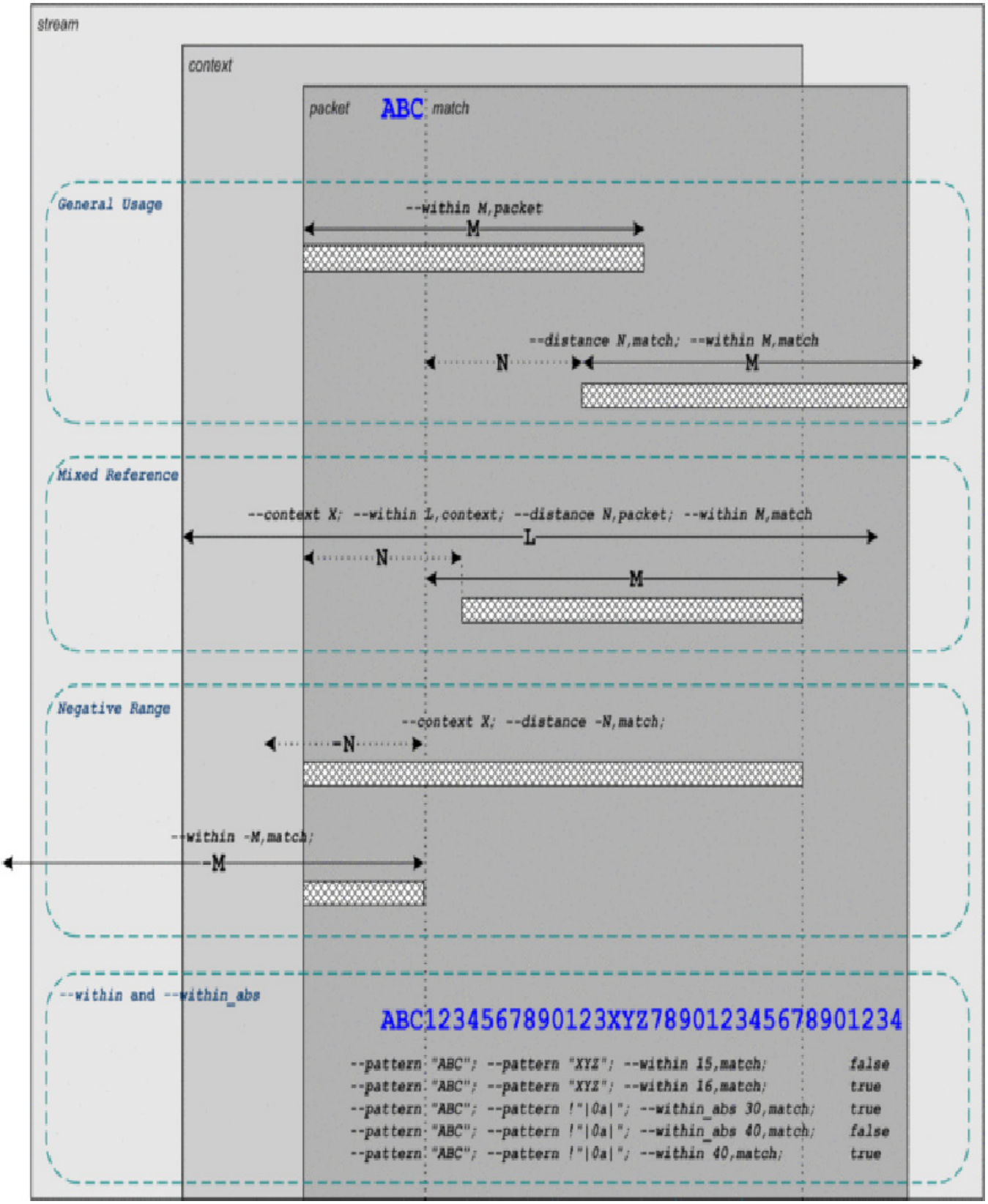



To check the website the client is trying to connect to in a HTTPS session, you could do this:

```
--service SSL; --pattern "api.sugarsync.com"; --context host;
     --no_case;
--service SSL; --pattern "simfy.de"; --context host; --no_case;
```

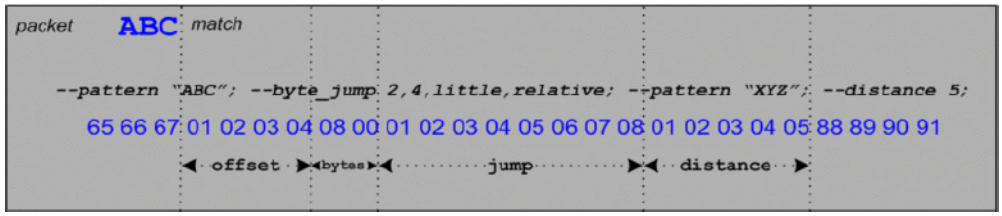
## Distance within diagram

**Figure 15:**Distance within



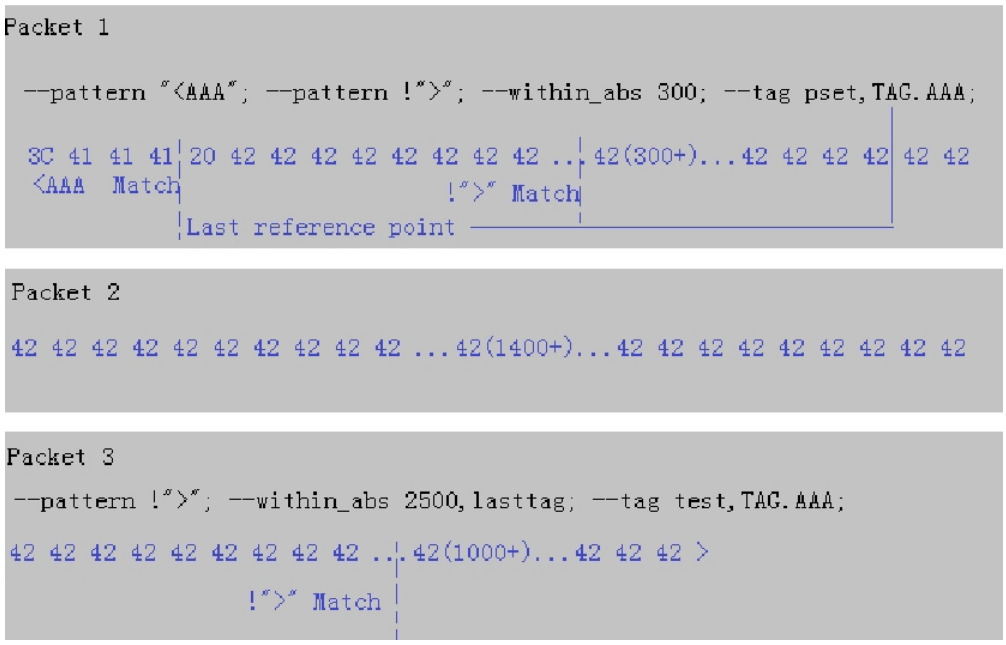
## Byte_jump diagram

**Figure 16:**Byte_jump




## PSET/LASTTAG diagram

**Figure 17:**PSET/LASTTAG



# Appendix C: Signature Definitions

## Context protocol notes

- The old content, uri, header and body keywords are deprecated. They are still supported in IPS Engine in order to keep backward compatibility, but should not to be used.
- The `U`, `H` and `B` modifiers have been removed from PCRE. Use `--context` to specify these conditions for PCRE matching, as shown in the table below. This is informational only.

  **Table 10:** Removed PCRE modifiers

| PCRE (Snort) | Description | Context |
|---|---|---|
| `U` | Match the decoded URI buffers | `uri` |
| `H` | Match normalized HTTP header | `header` |
| `B` | Do not use the decoded buffers | `packet_origin` |

- MIME parsing is supported for the email protocols SMTP, IMAP, POP3 and NNTP. Currently, we consider all attachments as falling into "`--context file`". We decode most of the encoding methods, including base64, uuencode, 7/8bit, quota, binary, and quoted-printable.
- For email protocols, some signatures used to use "`--context body`" for base64 encoded strings in attachments. They will not work anymore, because base64 is now decoded and the strings are located in the file context.
- For email protocols, the content that is not an attachment but is located in the body context, is considered as being in the body context.
- There is no body context in FTP, so the file context should be used.
- For HTTP, the body context and the file context are same. We can use either "`--context file`" or "`--context body`" to match the pattern.
- As an example, a signature like this should be able to match the *eicar* file in all of these protocols as it does not specify a service context and it uses file context:

  ```
  F-SBID( --protocol tcp; --pattern "X5O!P"; --context FILE; )
  ```

- If the file itself is zipped or archived, the engine currently does NOT decompress it.

## Range modifier notes

- The `offset`, `depth`, `offset_abs`, and `depth_abs` keywords are deprecated. These keywords are still supported in order to keep backward compatibility, but please do not use them to define official signatures.
- The Snort/PCRE `R` option has been removed from our PCRE. Please use `--distance 0;` instead.
- If no range modifier is used with `pattern`/`pcre`, matching is done from the beginning to the end of the buffer.
- If only `distance` or `distance_abs` is used with `pattern`/`pcre`, matching is done from the location that is relative to the reference specified by `<refer>` to the end of the buffer.
- If only `within` or `within_abs` is used with `pattern`/`pcre`, matching is done from the beginning of the reference specified by `<refer>`.

## Typical signature definition errors

- Doing the first pattern search (`--pattern`/`--pcre`) with one of the range modifiers (`distance`, `within`, `distance_abs`, or `within_abs`) for relative matching, with no `<refer>` value or with `<refer>` set to match. Since there was no previous match, no match reference point has been set.
- The `no_case` option should not be used on a nonalphabetic pattern. This is meaningless and has a negative effect on performance.
- The `no_case` option should not be used on casesensitive patterns. For example, some programming languages are casesensitive like C, Perl, PHP etc. Parameters passed to these programs should not use no_case.
- Signatures that set a `--tag` should have "`--status hidden;`". Signatures that set a Tag should not generate log entries or be shown in the GUI, so "`--status hidden;`" must be added. "`--status hidden;`" is used in addition to "`--tag quiet;`".
- Signatures that set a `--tag` should not be disabled in the default setting. This is because multiple signatures may depend on them.
- Searching for an encoded pattern in a URI. Multiple encoding methods can be used in a URI, and covering all encodings is not usually possible. In order to have better coverage, you should try to match the decoded pattern whenever possible.

  For example, you should use "`--pattern "/../../"; --context uri;`" rather than "`--pattern "/..%c0%af../"; context uri;`".
- Using special characters that are not escaped in PCRE can cause issues with detection. Since some characters have special meanings, they should be escaped by a backslash "`\`", or be expressed in hexadecimal format, like using `\x2e` for "`.`".
- Using the port number instead of `--service`. The engine can identify some well known services. Use these service types instead of their port number to define signatures. See "Service option types" on page 48 for the available service types.
- Using `byte_test` or `byte_jump` to do relative matching after a previous uri match. Since the HTTP decoder has three buffers for URIs, relative matching can cause the engine to panic resulting in performance issues.
- Wrongly combining `distance`, `within`, `distance_abs`, and `within_abs` for the same `pattern`/`pcre`. They should be used in pairs of `distance`/`within` and `distance_abs`/`within_abs`, and the `<refer>` values should be the same.

- Not specifying the best context value, or not including any context. Using patterns to match the URI, header, banner, host or body sectors of the traffic types HTTP, IMAP, SMTP, POP3 or SSH, without using the `--context` keyword, reduces efficiency and increases the possibility of false positives.

## Signature definition conventions

- All keywords in a signature are case-insensitive. Lower case is recommended.
- Patterns to be matched with `--pattern` must be enclosed in double quotes (") and followed by a semicolon (`;`). The special characters (`" ; \ | :`) must be written as (`|22|`, `|3B|` or `|3b|`, `|5C|` or `|5c|`, `|7C|` or `|7c|`, `|3A|` or `|3a|`). Although backslash (\) can be used to escape any character except a semicolon (`;`), its usage is not recommended.
- Patterns to be matched with `--pcre` must be enclosed in double quotes (") and followed by a semicolon (`;`). The special characters (`" ; /`) must be written as (`\x22`, `\x3B` or `\x3b`, `\x2F` or `\x2f`). Regular expressions should conform to the PCRE standard.
- Searching for short patterns, with length < 4, is inefficient for the engine's matching algorithm. It can cause false positives easily because short patterns have a high hit rate. It can also cause false negatives, because the engine does not support recursive matching. So it is recommended to use as long a pattern as possible with a range limit.
- Each rule should do at least 10 bytes of pattern matching to reduce false positives if possible.
- When searching for a pattern, a range limit should be specified, to reduce false positives as well as to improve performance.
- If some encoded content is always the same, a signature can be made to match the encoded form. This allows a signature to be made, even though the engine does not support decoding the content.
- For some file type exploits, you should check the file type tag first, before scanning. As an example, when scanning for an exploit in an Excel file, first test whether it is an Excel file before trying to search for a pattern.
- PCRE matching is the least efficient engine matching algorithm, and it can easily cause performance issues that are difficult to find by testing. So please take care when using PCRE.
- HTTP and Email are the predominant services on the Internet. Bad signatures for them causes the most complaints and lead to the worst after effects. Please take care when defining signatures for them.
- Use “`|0A|`”, not “`|0D 0A|`” to detect line endings, for example when scanning HTTP headers.